IFW

PTO/SB/21 (08-00)
Approved for use through 10/31/02. OMB 0651-0031
Patent and Trademark Office: U.S. DEPARTMENT OF COMMENRCE
Under the Paper Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays avalid OMB control number.

TRANSMITTAL FORM

(to be used for all correspondence after initial filing)

Application Number	10/626,120
Filing Date	July 23, 2003
First Named Inventor	Pierre Lebee, et al.
Group Art Unit	2857
Examiner Name	NYA
Total Number of Pages in This Submission: 21	Attorney Docket Number: 15437-0619

ENCLOSURES ***(check all that apply)***

- ☐ Fee Transmittal Form
  - ☐ Fee Attached
- ☐ Amendment / Response
  - ☐ After Final
  - ☐ Affidavits/declaration(s)
- ☐ Extension of Time Request
- ☐ Express Abandonment Request
- ☐ Information Disclosure Statement
- ☒ Certified Copy of Priority Document(s)
  from France dtd 24 Avr. 2003
- ☐ Response to Missing Parts/ Incomplete Application
  - ☐ Response to Missing Parts under 37 CFR 1.52 or 1.53
- ☐ Assignment Papers *(for an Application)*
- ☐ Drawing(s)
- ☐ Licensing-related Papers
- ☐ Petition
- ☐ Petition To Convert To a Provisional Application
- ☐ Power of Attorney, Revocation Change of Correspondence Address
- ☐ Terminal Disclaimer
- ☐ Request for Refund
- ☐ CD, number of CD(s)_____
- ☐ After Allowance Communication to Group
- ☐ Appeal Communication to Board of Appeals and Interferences
- ☐ Appeal Communication to Group *(Appeal Notice, Brief, Reply Brief)*
- ☐ Proprietary Information
- ☐ Status Letter
- ☐ Other Enclosure(s) *(please identify below):*

Remarks

SIGNATURE OF APPLICANT, ATTORNEY, OR AGENT

Firm or Individual name	Hickman Palermo Truong & Becker LLP Craig G. Holmes
Signature	
Date	June 3, 2004

CERTIFICATE OF MAILING

I hereby certify that this correspondence is being deposited with the United States Postal Service with sufficient postage as first class: mail in an envelope addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on this date: 6/3/04

Type or printed name	Annette Jacobs		
Signature		Date	June 3, 2004

Burden Hour Statement: This form is estimated to take 0.2 hours to complete. Time will vary depending upon the needs of the individual case. Any comments on the amount of time your are required to complete this form should be sent to the Chief Information Officer, Patent and Trademark Office, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

P6098

R E P U B L I Q U E F R A N Ç A I S E

INPI
INSTITUT NATIONAL DE LA PROPRIETE INDUSTRIELLE

BREVET D'INVENTION

CERTIFICAT D'UTILITÉ - CERTIFICAT D'ADDITION

COPIE OFFICIELLE

Le Directeur général de l'Institut national de la propriété industrielle certifie que le document ci-annexé est la copie certifiée conforme d'une demande de titre de propriété industrielle déposée à l'Institut.

Fait à Paris, le 24 AVR. 2003

Pour le Directeur général de l'Institut
national de la propriété industrielle
Le Chef du Département des brevets

Martine PLANCHE

INSTITUT NATIONAL DE LA PROPRIETE INDUSTRIELLE
SIEGE
26 bis, rue de Saint Petersbourg
75800 PARIS cedex 08
Téléphone : 33 (0)1 53 04 53 04
Télécopie : 33 (0)1 53 04 45 23
www.inpi.fr

ETABLISSEMENT PUBLIC NATIONAL CREE PAR LA LOI N° 51-444 DU 19 AVRIL 1951

1er dépôt

INPI
INSTITUT NATIONAL DE LA PROPRIÉTÉ INDUSTRIELLE
26 bis, rue de Saint Pétersbourg
75800 Paris Cedex 08
Téléphone : 01 53 04 53 04 Télécopie : 01 42 94 86 54

BREVET D'INVENTION
CERTIFICAT D'UTILITÉ

Code de la propriété intellectuelle - Livre VI

cerfa N° 11354*01

REQUÊTE EN DÉLIVRANCE 1/2

Important ! Remplir impérativement la 2ème page.

Cet imprimé est à remplir lisiblement à l'encre noire 08 540 W /190600

Réservé à l'INPI

REMISE DES PIÈCES
DATE 23 JUIL 2002
LIEU 75 INPI PARIS
N° D'ENREGISTREMENT NATIONAL ATTRIBUÉ PAR L'INPI 0209344
DATE DE DÉPÔT ATTRIBUÉE PAR L'INPI 23 JUIL. 2002

1 NOM ET ADRESSE DU DEMANDEUR OU DU MANDATAIRE À QUI LA CORRESPONDANCE DOIT ÊTRE ADRESSÉE

CABINET NETTER
36 avenue Hoche
75008 PARIS

Vos références pour ce dossier
(facultatif) SUN 21 (120590)

Confirmation d'un dépôt par télécopie ☐ N° attribué par l'INPI à la télécopie

2 NATURE DE LA DEMANDE	Cochez l'une des 4 cases suivantes
Demande de brevet	☒
Demande de certificat d'utilité	☐
Demande divisionnaire	☐
Demande de brevet initiale	N° Date / /
ou demande de certificat d'utilité initiale	N° Date / /
Transformation d'une demande de brevet européen *Demande de brevet initiale*	☐ N° Date / /

3 TITRE DE L'INVENTION (200 caractères ou espaces maximum)

Stress testing at low cost through parallel execution of unit tests.

4 DÉCLARATION DE PRIORITÉ OU REQUÊTE DU BÉNÉFICE DE LA DATE DE DÉPÔT D'UNE DEMANDE ANTÉRIEURE FRANÇAISE	
	Pays ou organisation Date / / N°
	Pays ou organisation Date / / N°
	Pays ou organisation Date / / N°
	☐ **S'il y a d'autres priorités, cochez la case et utilisez l'imprimé «Suite»**

5 DEMANDEUR		☐ **S'il y a d'autres demandeurs, cochez la case et utilisez l'imprimé «Suite»**
Nom ou dénomination sociale		SUN MICROSYSTEMS, INC
Prénoms		
Forme juridique		
N° SIREN		
Code APE-NAF		
Adresse	Rue	901 San Antonio Road
	Code postal et ville	94303 PALO ALTO Californie
Pays		Etats-Unis d'Amérique
Nationalité		Société des Etats-Unis d'Amérique
N° de téléphone *(facultatif)*		
N° de télécopie *(facultatif)*		
Adresse électronique *(facultatif)*		

operating system may be e.g. the POSIX interface used on UNIX based systems. The group of tests 5 comprises a set of tests such as test-1 to test-N referenced as 51 and 56. The test framework 4 comprises a test launcher 42 and test tools 41. The test framework may be seen as a software framework.

In figure 3, the test launcher is adapted to invoke the operating system to create a set of active entities 6 in memory (e.g. in RAM) comprising a set of processes such as process 1 to process j referenced as 61 to 66. Each process may be seen as a program instance. Each process comprises one thread at its creation, which may be called main thread. A thread may be seen as a program state instance. The test launcher is also adapted to invoke the operating system to create further threads in memory (e.g. in RAM) for each process. Figure 3 illustrates process 1 having i threads T1,1 to Ti,1 referenced as 611 and 612 and process j having i threads T1,j to Ti,j referenced as 661 and 662.

Figure 4 shows a more functional view of figure 3. Thus, the test launcher comprises a process and thread creation module 43. This module 43 is adapted, responsive to parameters defining the number of processes and corresponding threads, to invoke the operating system to create the processes 1, 2 to j referenced as 61, 62, 66 and their corresponding threads T1,1; Ti,1; T1,2; Ti,2; T1,j; Ti,j referenced as 611, 612, 621, 622, 661, 662.

The module 43 may further invoke the operating system to create supplementary process(es) and/or threads on an already created process's invocation.

The test launcher also comprises a process and thread deletion module 44. This module 44 is adapted:

- responsive to a deletion request coming from the main threads of each process, to invoke the operating system to delete threads of corresponding processes;
- to invoke for a deletion of a given process to the operating system.

To test the operating system, for instance in stressing conditions, a number of processes and corresponding threads are created. Test-1, test- 2, test- 3, test- 4, test- 5 to test- N referenced as tests 51, 52, 53, 54, 55 and 56 are applied repetitively, e.g. during L loops, and in parallel in each process and in each thread of the processes. At a given instant, a first thread may not

Stress testing at low cost through parallel execution of unit tests

This invention generally relates to the computer software technology, more precisely to execution of tests on operating systems.

Robust operating systems require to be tested, e.g. to detect problems in execution on operating systems, or to improve configuration of the operating systems. Thus, operating systems are nowadays tested with validation tests executed sequentially.

Such sequential execution does not permit to test operating systems in exceptional conditions, e.g. in stressing conditions. Thus, limits of the operating systems can only be determined partially and inefficiently.

A massive parallel execution of unit tests enables to simulate an overhead of test executions. However, such massive parallel execution of unit tests is time and resource consuming and/-or necessitates the design of a dedicated framework enabling such parallel executions.

A general aim of the present invention is to provide advances with respect to such mechanisms.

The invention concerns a method of testing an operating system comprising the steps of:
a. invoking the operating system to create a process, said process having a first thread,
b. executing repetitively at least a test function in said first thread of process,
c. repeating step a. and step b. so as to create processes and to execute the test function in parallel and repetitively in the first thread of created processes until a given condition is satisfied, said given condition comprising the operating system resources being exhausted.

The invention also concerns a framework for testing an operating system, said framework working in relation with at least a test function. This framework comprises a test launcher capable of
- performing invocation of the operating system to create a process, said process having a first thread,

- performing execution of the test function in said first thread of process repetitively, and capable of repeating said invocation and execution so as the test launcher is capable of performing execution of the test function in parallel and repetitively in the first thread of created processes until a given condition is satisfied, said given condition comprising the operating system resources being exhausted.

Other alternative features and advantages of the invention will appear in the detailed description below and in the appended drawings, in which :

- figure 1 is a general diagram of a computer system in which the invention is applicable;

- figure 2 is a portion of the diagram of figure 1 in more detail, in relation with a test framework and a group of tests according to the invention;

- figure 3 is a portion of the diagram of figure 2 in more detail, in relation with a set of active entities 6 according to the invention;

- figure 4 is a portion of the diagram of figure 3 in more detail, in relation with modules of the test framework;

- figure 5 is a flow chart for executing tests in parallel according to the invention;

- figure 6 is a flow chart of an example of test applied to an operating system.

As they may be cited in this specification, Sun, Sun Microsystems, Solaris, ChorusOS are trademarks of Sun Microsystems, Inc. SPARC is a trademark of SPARC International, Inc.

This invention also encompass software code, especially when made available on any appropriate computer-readable medium. The expression "computer-readable medium" includes a storage medium such as magnetic or optic, as well as a transmission medium such as a digital or analog signal.

In figure 1, this invention may be implemented in a computer system, or in a network comprising computer systems. The hardware of such a computer system is for example as shown in Fig.1, where:

- 11 is a processor, e.g. an Ultra-Sparc;
- 12 is a program memory, e.g. an EPROM for BIOS, a RAM, or Flash memory, or any other suitable type of memory;
- 13 is a working memory, e.g. a RAM of any suitable technology (SDRAM for example);
- 14 is a mass memory, e.g. one or more hard disks;
- 15 is a display, e.g. a monitor;
- 16 is a user input device, e.g. a keyboard and/or mouse; and
- 21 is a network interface device connected to a communication medium 20, itself in communication with other computers. Network interface device 21 may be an Ethernet device, a serial line device, or an ATM device, inter alia. Medium 20 may be based on wire cables, fiber optics, or radio-communications, for example.

Data may be exchanged between the components of figure 1 through a bus system 10, schematically shown as a single bus for simplification of the drawing. As is known, bus systems may often include a processor bus, e.g. of the PCI type, connected via appropriate bridges to e.g. an ISA bus and/or an SCSI bus.

In the foregoing description, the variables as L, N, i, or j are integers.
A test, also called a test function, may be seen as a program to test or to check the behavior of an operating system.

Figure 2 and 3 illustrates an exemplary embodiment. A processor 11 is adapted to work with an operating system, e.g. UNIX. To test this operating system, a test framework 4 is adapted to work on the basis of a group of tests 5. This test framework comprises a generic environment to execute tests. This generic test environment for executing tests on an

operating system may be e.g. the POSIX interface used on UNIX based systems. The group of tests 5 comprises a set of tests such as test-1 to test-N referenced as 51 and 56. The test framework 4 comprises a test launcher 42 and test tools 41. The test framework may be seen as a software framework.

In figure 3, the test launcher is adapted to invoke the operating system to create a set of active entities 6 in memory (e.g. in RAM) comprising a set of processes such as process 1 to process j referenced as 61 to 66. Each process may be seen as a program instance. Each process comprises one thread at its creation, which may be called main thread. A thread may be seen as a program state instance. The test launcher is also adapted to invoke the operating system to create further threads in memory (e.g. in RAM) for each process. Figure 3 illustrates process 1 having i threads T1,1 to Ti,1 referenced as 611 and 612 and process j having i threads T1,j to Ti,j referenced as 661 and 662.

Figure 4 shows a more functional view of figure 3. Thus, the test launcher comprises a process and thread creation module 43. This module 43 is adapted, responsive to parameters defining the number of processes and corresponding threads, to invoke the operating system to create the processes 1, 2 to j referenced as 61, 62, 66 and their corresponding threads T1,1; Ti,1; T1,2; Ti,2; T1,j; Ti,j referenced as 611, 612, 621, 622, 661, 662.

The module 43 may further invoke the operating system to create supplementary process(es) and/or threads on an already created process's invocation.

The test launcher also comprises a process and thread deletion module 44. This module 44 is adapted:

- responsive to a deletion request coming from the main threads of each process, to invoke the operating system to delete threads of corresponding processes;
- to invoke for a deletion of a given process to the operating system.

To test the operating system, for instance in stressing conditions, a number of processes and corresponding threads are created. Test-1, test- 2, test- 3, test- 4, test- 5 to test- N referenced as tests 51, 52, 53, 54, 55 and 56 are applied repetitively, e.g. during L loops, and in parallel in each process and in each thread of the processes. At a given instant, a first thread may not

execute the same test as a second thread. In other words, the instant of execution of a given test may be different in each thread.

In a particular embodiment, the tests invoked by each thread may be chosen randomly amongst the N tests.

Figure 5 illustrates a method 100 for executing tests in parallel in different threads. Thus, at operation 101, a user, who wishes to test the operating system, may choose parameters and send them to the test framework. These parameters may comprise the number of processes to be created. These parameters may also comprise the number of threads to be created. These parameters may be stored in advance, e.g. in a configuration file. Thus, at operation 102, the user may invoke the process and thread creation module to firstly create the number of required processes. To create this required number of processes, the process and thread creation module invokes the operating system. Each process is created with a main thread.

At operation 104, each main thread in each process may invoke a test initialization. Then, at operation 106, if the user has invoked creation of other threads at operation 101, the process and thread creation module permits the creation of the required number of threads in each process. The test tools module comprises tools to detect problems such as creation problems, e.g. these tools may comprise sending error messages.

Thus, at operation 108, according to the number of processes and threads, the test launcher manages the execution of tests through the exemplary following execution modes :

-mono-thread (i=1) and mono-process mode (j=1);
in this first mode, the tests are executed by a single thread in a single process;
-multi-thread (i>1) and mono-process mode (j=1);
in this second mode, the tests are executed in parallel by multiple threads in a single process;
- mono-thread (i=1) and multi-process mode (j>1);
in this third mode, the tests are executed in parallel by a single thread in multiple processes;
- multi-thread (i>1) and multi-process mode (j>1);
in this fourth mode, the tests are executed in parallel by multiple threads in multiple processes.

The first mode authorizes a repetitive sequential execution of a set of tests on the single thread of the single process. Even after a long time of execution using this first mode, a problem concerning the operating system, e.g. resource missing, may not appear. Thus, to provide a more complete and faster error detection, the other modes authorize a repetitive execution of a set of tests on each thread of each process, called parallel execution.

In another embodiment, operations 101 to 108 may be applied for one process having at least one thread. Then, these operations may be repeated to create in parallel other processes having at least one thread. These creation operations may be applied to overload the operating system until operating system resources are exhausted. The operating system has then no more resources to create other processes and/or threads. These creation operations may also be applied until reaching a selected number of process and/or threads.

In the foregoing description, a test may be divided into several actions called portions of test. For an operating system, executing tests in parallel in different threads may mean executing tests or portions of tests sequentially in different threads according to thread priority and time-sharing. In a first execution example, portions of an identical test may be authorized to be executed in different threads alternatively and in parallel. In a second execution example, an identical test may be authorized to be executed entirely only in one thread before being executed in another thread.

In addition, the test launcher makes every thread of every processes sequentially execute repetitively the different tests which are embedded in the group of tests. Tests are adapted to use primitives to be called and global variables to be read, both provided by the test tools module for executing tests in parallel. For example, primitives may enable at the test level to check if allocated memory for tests has been released, or if files opened during execution of tests have been closed.

Tests may follow basic rules to be embedded into the test framework using e.g. the POSIX interface. As an example, the list below describes some of these conditions :

a- tests may be aware of their own execution in parallel, execution realized in multiple active entities (threads/processes)

b- point a- implies that tests may assign unique names to all resources they dynamically create. For instance, to guaranty a unique name for a file corresponding to a given thread in a given process, the file name may be composed with the thread and process identifier, e.g. */foo_th_23_proc_67* for thread 23 and process 67. This rule may be comprised in the rules of the test tools module

c- point a- may also implies that data used by tests may be private, e.g. automatically allocated or dynamically allocated. To access to global variables, mutual exclusion is needed.

When launching a test program, three arguments may be provided to the test framework:

- i, the number of processes
- j, the number of threads within every process
- L, the number of times every thread of every process will execute the N different tests.

By providing appropriate values for these three numbers, a given set of tests can be used to heavily stress an operating system by having the tests executed with as much parallelism as desired or required.

In an embodiment, the user may choose the number j of processes to be created, the number i of threads to be created, which may be different for each process, and the number of loops L to apply tests to each thread of each process in parallel. In this embodiment, the operating system may be tested in normal conditions or in more stressing conditions according to the choice of the user. In another embodiment, the choice of the variables j, i and L may be automatically provided by the test launcher 42.

More generally, the load, which can be exercised on an operating system with the same set of tests, may be seen as unlimited when using the parallel execution of tests in each thread of processes.

At operation 112, each main thread of each process may delete the other threads of processes. Then, each main thread may invoke the test termination at operation 114. Each process may be deleted at operation 116.

In figure 6, an example of test is represented. At operation 202, the required result is to create a file, which may be done according to given rules in the test tools. At operation 204, if the result obtained is not the one required, an error called error 1 is sent to the framework at operation 206 and the test ends at operation 215. Else, the test continue at operation 208.

The required result is to delete the created file. At operation 208, if the result obtained is not the one required, an error called error 2 is sent to the framework at operation 212 and the test ends at operation 215. Else, a message "OK" is sent to the framework at operation 214 and the test ends at operation 215.

In the case of a message "OK", the test is successful for the operating system. In the other cases, e.g. in operation 206 or 212, the type of error may determine if the operating system has failed or if other problems appeared (e.g. resource missing).

The test framework which creates the set of active entities 6 may be comprised in a product being a software package to test systems, e.g. operating systems.

The invention is not limited to the hereinabove described features.

Thus, other interfaces than POSIX may be used to execute tests on other systems.

The structure comprising the test framework interacting with a group of tests and a set of active entities may test an operating system working in relation with one or more processors. Moreover, this structure may test data base server working in relation with one or more processors. In another embodiment, this structure may test java virtual machine (JVM) comprising threads working in relation with one or more processors.

Claims

1. A method of testing an operating system comprising the steps of:

a. invoking the operating system to create a process (101), said process having a first thread,

b. executing repetitively at least a test function in said first thread of process (108),

c. repeating step a. and step b. so as to create processes and to execute the test function in parallel and repetitively in the first thread of created processes until a given condition is satisfied, said given condition comprising the operating system resources being exhausted.

2. The method of claim 1, wherein step b. further comprises initializing at least the test function.

3. The method of claim 1, wherein the given condition of step c. further comprises a selected number of processes.

4. The method of claim 1, wherein the selected number of processes of step c. further comprises a number of processes chosen by the user.

5. The method of claim 1, wherein step a. further comprises invoking the operating system to create at least a second thread in said process.

6. The method of claim 5, wherein said second thread of step a. is created by the operating system in relation to a configuration file.

7. The method of claim 5, wherein said second thread of step a. is created by the operating system upon user request.

8. The method of claim 5, wherein said second thread of step a. is created by the operating system upon first thread request.

9. The method as claimed in any of claims 5 through 8, wherein step b. further comprises executing repetitively a test function in the first and at least in the second thread of said process.

α

10. The method as claimed in any of claims 5 through 9, wherein step c. comprises repeating step a. and step b. so as to create processes with first and at least second threads and to execute the test function in parallel and repetitively in the first and at least second thread of created processes until the given condition is satisfied.

11. The method as claimed in any of claims 5 through 10, wherein step c. further comprises c1. retrieving results of the test function applied in parallel in first and at least second threads of created processes.

12. The method of claim 11, wherein step c. further comprises
c2. deleting said at least second thread of created processes,
c3. invoking test function termination by first thread of created processes.

13. The method of claim 12, wherein step c. further comprises deleting said created processes.

14. A framework (4) for testing an operating system (2), said framework (4) working in relation with at least a test function (51), wherein said framework comprising a test launcher (42) capable of
- performing invocation of the operating system (2) to create a process (61), said process (61) having a first thread (611),
- performing execution of the test function (51) in said first thread (611) of process (61) repetitively,
and capable of repeating said invocation and execution so as the test launcher is capable of performing execution of the test function (5) in parallel and repetitively in the first thread of created processes (61,66) until a given condition is satisfied, said given condition comprising the operating system resources being exhausted.

15. The framework of claim 14, wherein the test launcher (42) is further capable of initializing at least the test function (51).

16. The framework of claim 14, wherein the given condition comprises a selected number of processes.

17. The framework of claim 16, wherein the selected number of processes comprises a number of processes chosen by the user.

18. The framework as claimed in any of claims 14 through 17, wherein the test launcher (42) is further capable of performing invocation of the operating system (2) to create at least a second thread (612) in said process (61) .

19. The framework of claim 18, wherein said second thread is created by the operating system in relation to a configuration file.

20. The framework of claim 18, wherein said second thread is created by the operating system upon user request.

21. The framework of claim 18, wherein said second thread is created by the operating system upon first thread request.

22. The framework as claimed in any of claims 18 through 21, wherein the test launcher (42) is further capable of performing execution repetitively and in parallel of the test function (5) in the first (611) and second thread (612) of said created processes (61,66) until the given condition is satisfied.

23. The framework as claimed in any of claims 18 through 22, wherein the test launcher (42) is further adapted to retrieve results of the test function applied in parallel in first and at least second thread of created processes.

24. The framework of claim 23, wherein the test launcher (42) comprises a deletion module (44) adapted to delete said at least second thread of created processes, and to invoke test function termination by first thread of created processes.

25. The framework of claim 24, wherein the test launcher (42) comprises a deletion of said created processes.

26. A software product, comprising the software functions for use in the method of testing the operating system in any of claims 1 through 13.

27. A software product, comprising the software functions used in the framework for testing an operating system as claimed in any of claims 14 through 25.

α (12 pages)

CABINET NETTER

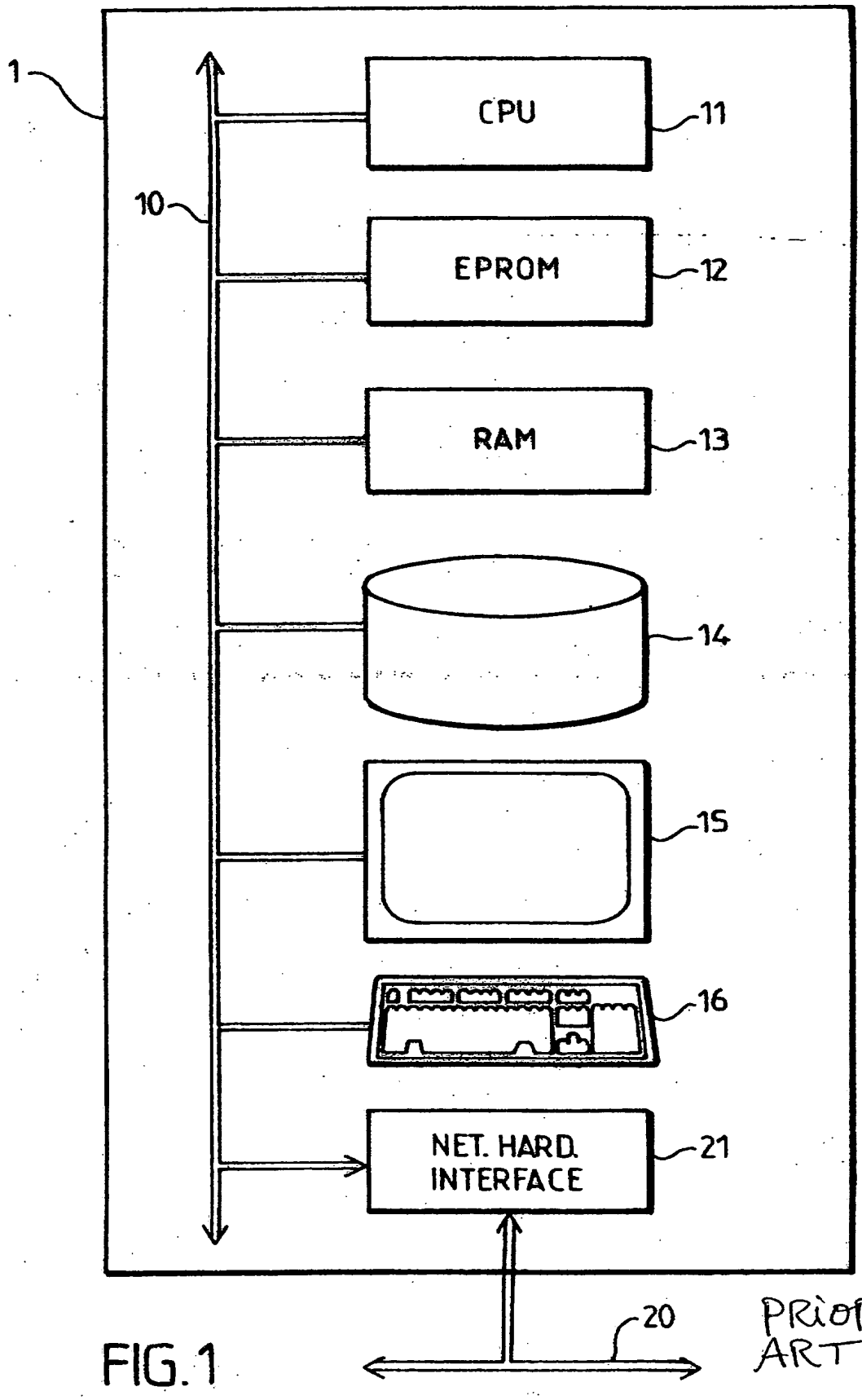


FIG.1

PRIOR ART

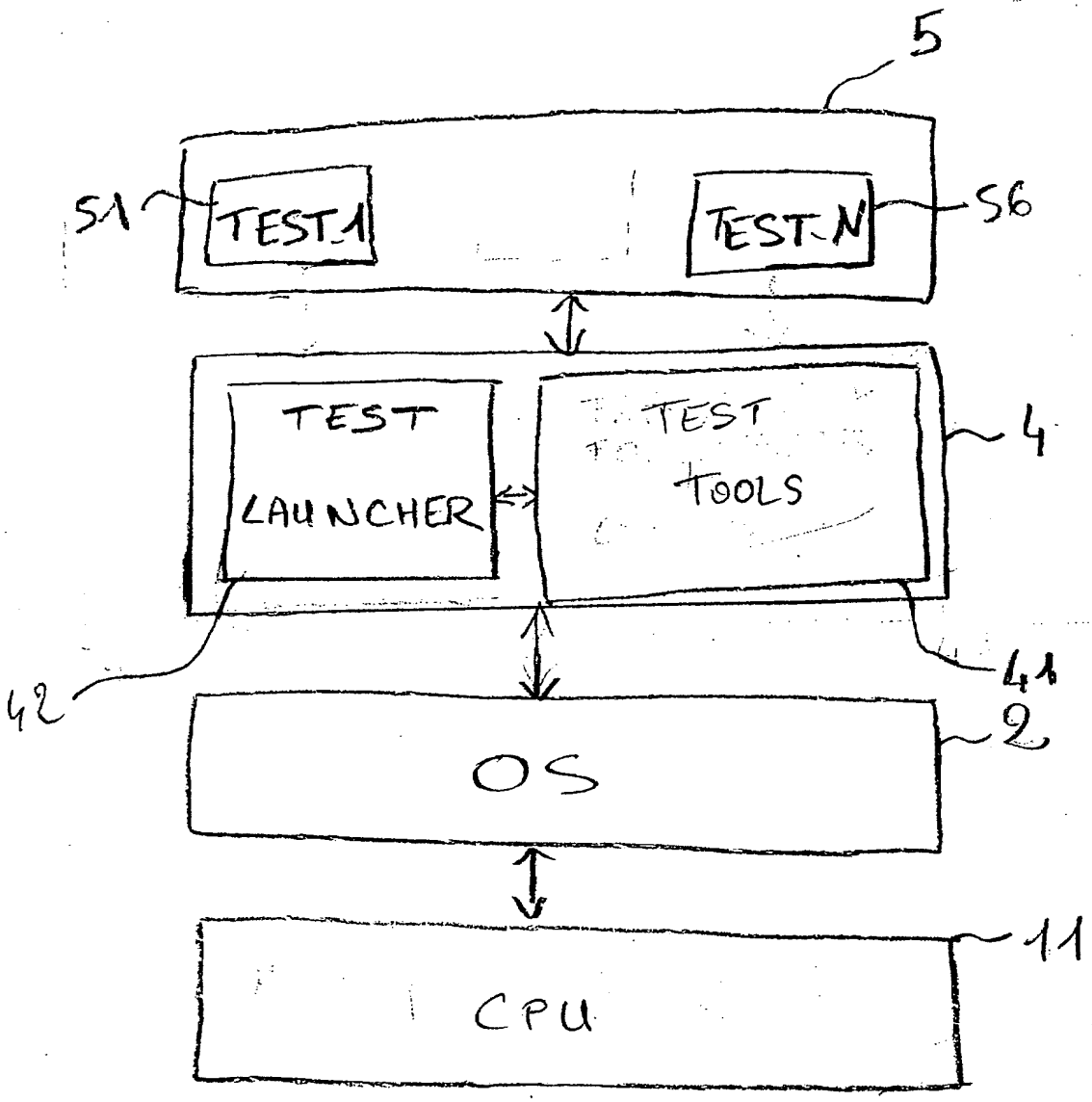
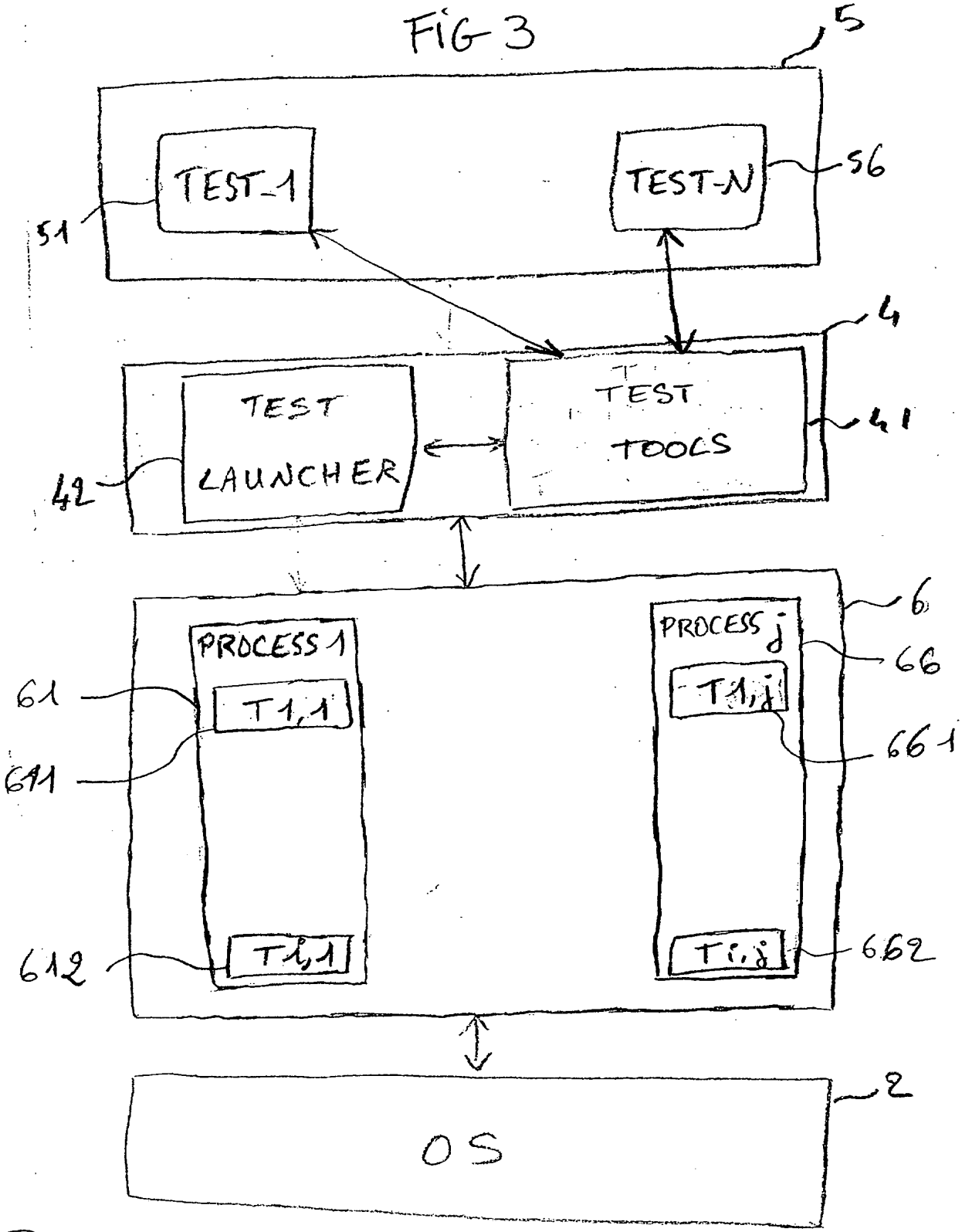


FIG 2

FIG 3


5
TEST_1
TEST-N
51
56
4
TEST
LAUNCHER
TEST
TOOLS
42
41
6
PROCESS 1
PROCESS j
61
66
T1,1
T1,j
611
661
612
Ti,1
Ti,j
662
2
OS

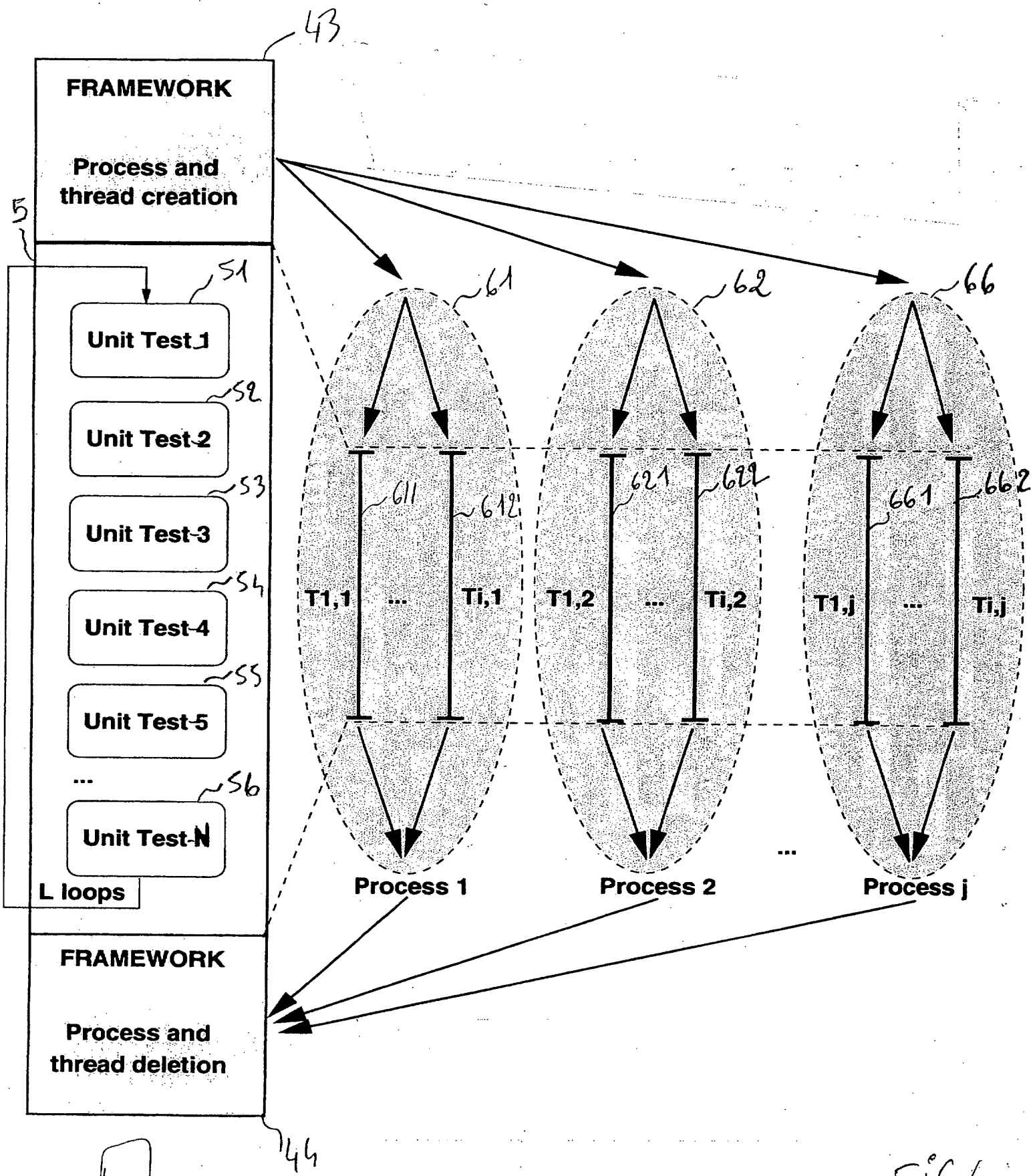

43
FRAMEWORK
Process and thread creation
5
51
Unit Test-1
52
Unit Test-2
53
Unit Test-3
54
Unit Test-4
55
Unit Test-5
...
56
Unit Test-N
L loops
61
62
66
611
612
621
622
661
662
T1,1
...
Ti,1
T1,2
...
Ti,2
T1,j
...
Ti,j
Process 1
Process 2
Process j
FRAMEWORK
Process and thread deletion
44

FIG 4

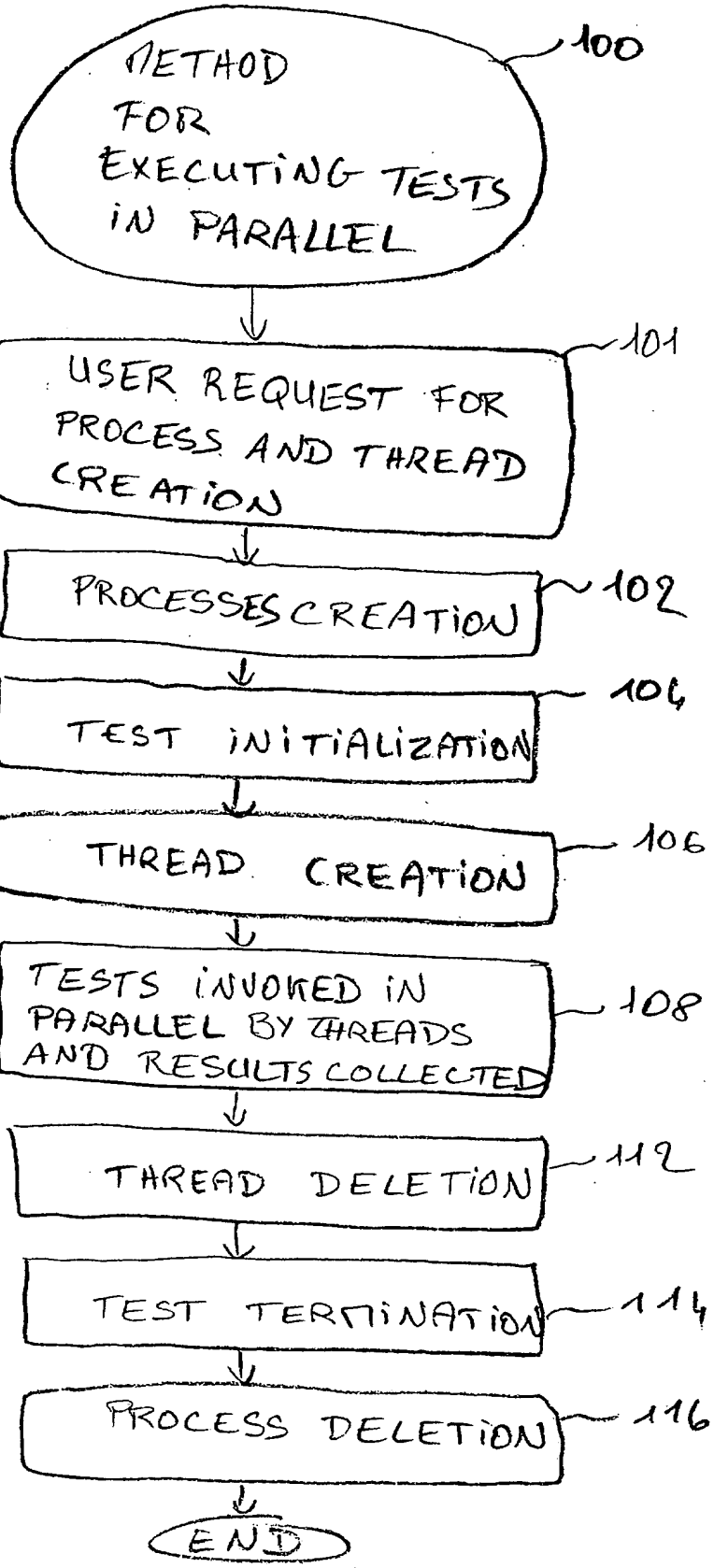

METHOD FOR EXECUTING TESTS IN PARALLEL
100
USER REQUEST FOR PROCESS AND THREAD CREATION
101
PROCESSES CREATION
102
TEST INITIALIZATION
104
THREAD CREATION
106
TESTS INVOKED IN PARALLEL BY THREADS AND RESULTS COLLECTED
108
THREAD DELETION
112
TEST TERMINATION
114
PROCESS DELETION
116
END

FIG-5

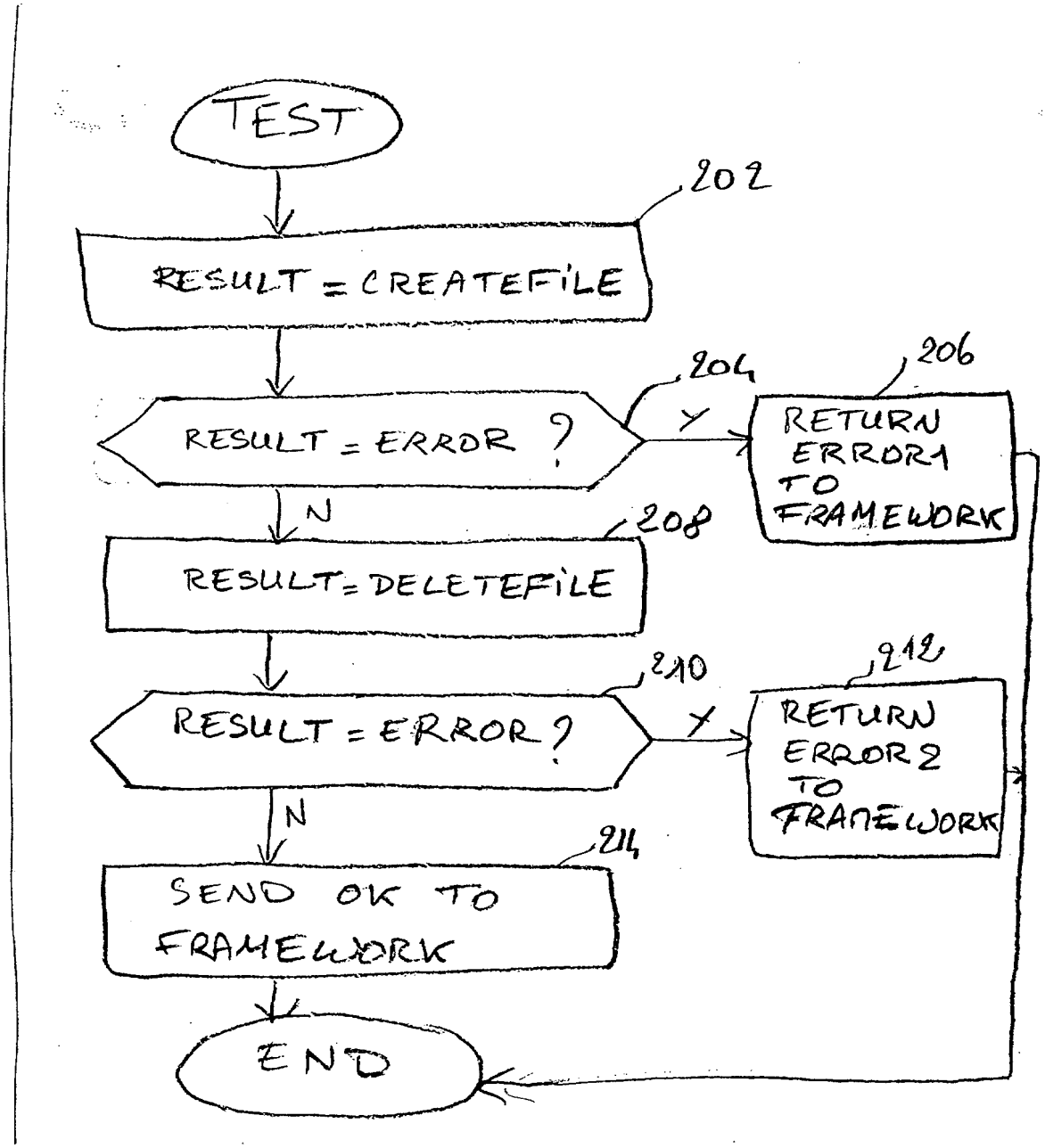


FIG-6